REMARKS

Claims 1-28 are now pending in this application. Claims 1, 14, 19 and 24 are objected to for minor informalities. Claims 1, 3-12, 14, 16-17, 20-22, 24 and 26-35 stand rejected under 35 U.S.C. § 102(e) as allegedly being anticipated by United States Patent Number 6,922,708 ("Sedlar"). Claims 2, 15, 19 and 25 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Sedlar in view of United States Patent Number 7,035,874 B1 ("Reed"). Applicants respectfully traverse.

Claims 1-6, 14,19 and 24 have been amended.

Objection To Claim Informalities

Claims 1, 14, 19 and 24 are objected to for minor informalities. These claims have been amended and now overcome the objections raised by the Examiner.

Claims 1-12

Claims 1 and 3-12 stand rejected under 35 U.S.C. § 102(e) as allegedly being anticipated by Sedlar. Claim 2 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over Sedlar in view of Reed.

Claim 1 as amended recites a method for executing a transaction comprising at least one query language statement and at least one file system statement, each statement relating to a user define type ("UDT"), comprising receiving each of the at least one file system statement, *wherein each item referenced in a statement is stored separately from a database table associated with the item*, *receiving each of the at last one query language statement*, *creating a transaction as part of an open operation, wherein the transaction is managed separately from the database server* and for each query language statement, *starting a transaction on the database server updating fields associated with the item in the query language statement and sending an updategram to the database server*. Support for this amendment may be found, for example, on pages 8-14 of the specification.

Briefly summarizing this discussion, Applicant's application is directed to a method and system for unifying a sharing model for file system operations with a transaction and locking model for query language statements to provide an overall framework for locking and

isolation of filestreams in a storage platform. Unlike file system statements, which are only processed for out of band access to file stream it items, query language statements may be executed upon both file stream and non-file stream items. In addition, transactional support is provided for file system operations so that they may be executed in the context of a transaction. Accordingly, a single transaction may include any number or combination of file system statements and query language statements. Support is also provided for non-transacted file system operations so that file system operations need to necessarily be executed in the context of a transaction.

Referring to Fig. 1 of the application, instances of storage platform items may be stored in a relational database table 200 which may include two columns: an ID column 202a and user defined type ("UDT") column 202b. Each ID in the ID column 202a provides a unique identifier for a corresponding UDT. Each UDT in the UDT column 202b corresponds to an instance of an item type and may be filestream fields, meaning that data for that field is stored separately from the table 200.

Referring to Fig. 2, which shows an exemplary storage platform, a client application 300 may manipulate items in a data store 310 directly through a storage platform 308 or via "out of band" access through a file system API 302. If the client application 300 manipulates items directly through the storage platform 308, such manipulation may be performed using a query language such as T-SQL. If, on the other hand, the client application manipulates items via "out of band" access through the file system API 302, such manipulation may be performed using a file system API such as Win32. "Out of band" access may only be available for items that include filestream fields.

A client application may access the storage platform directly by using storage platform methods to initiate a query on the datastore. The storage platform translates an operation into a query in T-SQL or another query language and submits it to an underlying data store. The data store then executes the query against corresponding instances of a UDT and returns stored values for each matching instance of the UDT. The storage platform then wraps the UDP objects and returns them to the application.

To unify the transactional model of the query language with the sharing model of the file system, it is necessary to define the concept of a file system statement. Such a file system statement may include an open operation, either a read or a write operation, and a close

operation. Thus, a file system statement that includes open, read, and close operations is semantically equivalent to a SELECT statement in the query language, while a file system statement that includes open, right and close operations is semantically equivalent to an UPDATE statement in the query language.

File system statements may occur in the context of a multi-statement transaction or a single statement transaction, in which the transaction is present during the processing of the statement. A transaction may include either a single query language statement, a single file system statement, multiple query language statements, multiple file system statements, or a combination of query language and file system statements.

In the case of a combination of query language and file system statements, a transaction is started on the server, and updategrams that modify non-filestream fields in an item are provided to a database server. File system statements may then be executed in the context of the transaction. The transaction is then committed on the server.

A transaction context may be specified for file system operations to obtain the appropriate locks on corresponding rows. If an open is for a read, then the transaction will obtain the read lock on the row containing the file stream. If an open is for a write, then the transaction will obtain a write lock on the row containing the file stream.

Referring to Fig. 3 of the application, an exemplary method for executing a file system statement in the context of the transaction is shown. At act 410, a storage platform 308 receives statements from a client application 300 to be executed upon items stored in a data store 310. Such statements may be query language statements received from the client application 300 or out of band file system statements received from a client application 300 the eighth file system API 802 and file system aged 804. Query language statements may be executed upon both filestream and non-filestream items while file system statements may only be executed upon filestream items.

A file system statement may include an open operation, either a reapplication or a write operation, and a close operation. Each file system statement may be submitted in the context of the transaction. An act 412, each statement received at act 410 is associated with the transaction. At act 414 it is determined whether starting the transaction will result in a conflict. For example, if any statements within the transaction correspond to a row upon which another transaction has already acquired a write lock, then a conflict will occur. If, at

act 414, it is determined that a conflict will occur, then an act 416, the conflict is resolved. At act 418, the transaction is started by acquiring read and write locks on the appropriate rows. Read locks are required on rows corresponding to read operations and to select statements.

Write locks are acquired on rows corresponding to write operations and to update statements. The write lock is an exclusive lock is acquired for the lifetime of the transaction. The write lock will prevent another transaction from accessing either through read access or write access a corresponding row while the transaction is being processed. The write lock will also prevent a non-transacted file system statement from accessing a corresponding row while the transaction is being processed.

To emulate the query language isolation model, an exclusive open may be obtained for writers, while a shared open may be obtained for readers. The exclusive and shared opens correspond respectively to an exclusive row level locking and a shared row level locking in the query language model. The sharing modes may provide isolation at two levels, namely between transactions in between opens within a transaction. In the storage platform, the unit of isolation maybe the item, an extension of associated with the item, or relationship associated with the item. In each of these three cases, the underlying representation is an instance of a UDT in an appropriate table, and the unit of isolation corresponds to row in the table. This maps to the query language locking model, in which the appropriate locks are obtained on a per row basis.

Sedlar describes a file system interface that is exposed to applications. The file system interface includes routines for saving and retrieving files. Calls to perform a plurality of file operations are received through the file system interface. The plurality of file operations are performed as a single transaction. Sedlar describes a method and system to allow the same set of data to be accessed through a variety of interfaces, including a database API and an OS file system API. A database server provides a database API through which a database application can access data managed by the database server through the database API. A translation engine is described that translates I/O commands received from an operating system into database commands that the translation engine issues up to the database server. When the I/O commands call for the storage of data, the translation engine issues database commands to the database server to cause a data to be stored in relational tables managed by the database server. When the I/O commands call for retrieval of data, the translation engine issues

database commands to the database server to retrieve data from the relational tables managed by the database server. The translation engine then provides the data thus retrieved to the operating system that issued the I/O commands.

To the operating system, the fact that data passed to a translation engine is ultimately stored in a relational database management database server is transparent. The process of performing these translations is performed by emulating within the database server the characteristics of the file systems implemented by the operating system. In particular a mechanism is described for emulating a hierarchical file system within a relational database system. Sedlar further describes an OS file API that may include routines to perform the operations of open file, read from file, write to file, seek within a file, lock a file and close file.

Sedlar fails to teach or suggest receiving each of at least one file system statement . . . *wherein each item in a statement is stored separately from a database table associated with an item*, *receiving each of at least one query language statement*, wherein each query language statement is associated with a item, *creating a transaction, wherein the transaction is managed separately from the database server* and *for each query language statement, starting a transaction on the database server updating fields associated with the query language statement and sending an updategram to the database server* as recited in claim 1. Sedlar fails to teach or describe a method whereby both file system statements and query language statements can be handled in a unified manner. Sedlar describes a translation engine that receives I/O operations and translates those into commands for providing to a DB file server. However, Sedlar fails to teach or suggest handling both file system statements and query language statements.

Furthermore, in the system described by Sedlar, a translation engine translates I/O commands from an operating system into database commands. However, ultimately any transactions flowing from these commands are necessarily managed by the database server (since the commands are merely translated) rather than another entity such as a platform server. Moreover, Sedlar fails to teach or suggest that each item referenced in a file system statement is stored separately from a database table associated with the item as required by amended claim 1.

As the cited reference fails to teach or suggest the claimed elements, claim 1 should be

allowed. Claims 3-12 depend from and therefore include all the limitations of claim 1. Thus, for at least the reasons stated with respect to claim 1, claims 3-12 should be allowed. In addition, as claim 3 includes all the limitations of claim 1 and Reed fails to cure the deficiencies of Sedlar, claim 3 should also be allowed.

Claims 14-17

Claim 14 as amended recites limitations similar to claim 1. In particular, claim 14 as amended recites receiving a file system statement comprising a call to open an item. . . *wherein each item referenced in the statement is stored separately from a database table associated with the item.* Thus, for at least the reasons stated with respect to claim 1, claim 14 should be allowed. Claims 15-17 depend from claim 14 and therefore include all the limitations of claim 14. Thus, for at least the reasons stated with respect to claim 14, claims 15-17 should be allowed.

Claims 19-22

Claim 19 as amended recites limitations similar to claim 1. In particular, claim 14 as amended recites a method for locking and isolation of a file system statement . . . comprising: receiving the file system statement *wherein each item referenced in the statement is stored separately from a database table associated with the item.* Thus, for at least the reasons stated with respect to claim 1, claim 19 should be allowed. Claims 20-22 depend from claim 19 and therefore include all the limitations of claim 19. Thus, for at least the reasons stated with respect to claim 19, claims 20-22 should be allowed.

Claims 24-35

Claim 24 as amended recites limitations similar to claim 1. In particular, claim 24 as amended recites a system for executing a file system statement . . . comprising a storage platform built on the relational data engine, the storage platform comprising means for receiving the file system statement, *wherein each item referenced in the statement is stored separately from a database table associated with the item.* Thus, for at least the reasons stated with respect to claim 1, claim 24 should be allowed. Claims 25-35 depend from claim

24 and therefore include all the limitations of claim 24. Thus, for at least the reasons stated with respect to claim 24, claims 25-35 should be allowed.

CONCLUSION

In view of the above amendments and remarks, applicant respectfully submits that the present invention is in condition for allowance. Reconsideration of the application is respectfully requested.

Date: September 15, 2008

/Kenneth R. Eiferman/
Kenneth R. Eiferman
Registration No. 51,647

Woodcock Washburn LLP
Cira Centre
2929 Arch Street, 12th Floor
Philadelphia, PA 19104-2891
Telephone: (215) 568-3100
Facsimile: (215) 568-3439